95-702 Distributed Systems

HTTP & Server-Side Programming

Prior Experience in Distributed Systems

- Please take the short survey at:
  - **http://tinyurl.com/95702prior**
- Don't do now, I will send the link to you after class

Goals

- Understand the HTTP application protocol
  - Request and response messages
  - Methods / safety / idempotence
- Understand Java server-side web application terminology
  - Servlet
  - jsp
  - Web container
  - HttpServlet
- Understand the flow of control
  - Web server, web container, servlet
- Understand flow of request to response
  - HttpServletRequest
  - HttpServletResponse
- Understand MVC in servlets and jsp.

HTTP

- HTTP is the application protocol with which most distributed systems communicate:
  - Browser communicating with web sites and web applications
    - When your browser contacts a web site/app, it uses the HTTP protocol (thus the http:// in the browser address bar)
    - When the server responds, it is responding using the HTTP protocol
  - Clients communicating with web services
- This is certainly not the only protocol, and we will be discussing others this semester, but in terms of traffic, HTTP is a critical protocol to know and understand.

HTTP Interaction Pattern

- Uses Request / Response interaction pattern
  - Client
    - Establishes a connection with the server
    - Sends a request message and waits
  - Server
    - Processes the request as it is received
    - Sends a response message over the same client connection
- Coupled in time:
  - Both client and server must be available at time of the interaction
- Coupled in space:
  - Both client and server must know the address of each other, and communicate directly point-to-point
- Can also be used for Request / Acknowledge

HTTP Request

General Format	Example

`<method> <resource identifier> <HTTP Version> <crlf>`
`[<Header>: <value>] <crlf>`
`...`
`[<Header>: <value>] <crlf>`
`   a blank line`
`[entity body]`
	`GET /course/95-702/ HTTP/1.1`
	`Host: www.andrew.cmu.edu`
	`User-Agent: Joe typing`
	`Accept: text/html`
	This line intentionally left blank

- Method
  - GET, PUT, DELETE, HEAD, POST, etc.
- Resource identifier specifies the name of the target resource;
  - i.e. it's the URL stripped of the protocol and the server domain name.
  - When using the GET method,
    - this field will also contain a series of name=value pairs separated by ‘&’.
  - When using a POST method,
    - the entity body contains these pairs.

- HTTP version identifies the protocol used by the client.

HTTP Response

General Format	Example

`<HTTP Version> <Status> <crlf>`
`[<Header>: <value>] <crlf>`
` ...`
`[<Header>: <value>] <crlf>`
`   a blank line`
`[response body]`
	`HTTP/1.1 200 OK`
	`Date: Mon, 13 Jan 2014 15:43:08 GMT`
	`Server: Apache/1.3.39 (Unix) mod_throttle/3.1.2 ...`
	`Set-Cookie: webstats-cmu=cmu128.2.87.50.8400; ...`
	`Last-Modified: Sun, 12 Jan 2014 21:46:30 GMT`
	`Accept-Ranges: bytes`
	`Content-Length: 9014`
	`Content-Type: text/html`
	This line intentionally left blank
	`<HTML>`
	`<HEAD>`
	`<META http-equiv="Content-Type" content="text/html; charset=UTF-8">`
	`...`

- HTTP version identifies the protocol used by the client.
- Status indicates the result of the request

HTTP Request with a Query String

```
GET /alflickrbet?letter=e HTTP/1.1
Host: newalflickrbet.appspot.com
User-Agent: Joe typing
Accept: text/html
```

HTTP Response Example

- If response length is not known, chunks of data are sent
- The size of the chunk is given, followed by that # of bytes
- 0x2d6 = 726 (I edited out some whitespace for the slide)
- There could be any number of chunks
- End of chunks is indicated by length 0
- Why is the length not known in this case?

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
Date: Tue, 14 Jan 2014 02:15:23 GMT
Server: Google Frontend
Cache-Control: private
Transfer-Encoding: chunked

2d6
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
      <title>alFLICKRbet</title>
  </head>
  <body>
    <h1> e is for elephant </h1><br>
    (Source: cached in datastore )<br><br>
    <a href=http://www.flickr.com/67955941@N00/3885234912 >
    <img src=http://farm3.static.flickr.com/2488/3885234912_5278906961_z.jpg>
    </a><form action="alflickrbet">
    <br><label for="letter">Type the letter you would like to play with next: </label>
    <input type="text" name="letter" value="" /><br>
    <input type="submit" value="Submit" /></form>
    <br>
  </body>
</html>
0
```

Some Common HTTP Status Codes

- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 404 Not found
- 500 Internal Server Error
  - You get this when your server program throws an uncaught exception.
  - You are likely to see this frequently this semester!

- For more information, see the standard:
  - http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Show example

- Show in Chrome
- Show via telnet
- Show in curl
  - MacOS: included
  - Windows: http://curl.haxx.se/download.html#Win32
- REST Console for Chrome
  - https://chrome.google.com/webstore/detail/rest-console/cokgbflfommojglbmbpenpphppikmonn

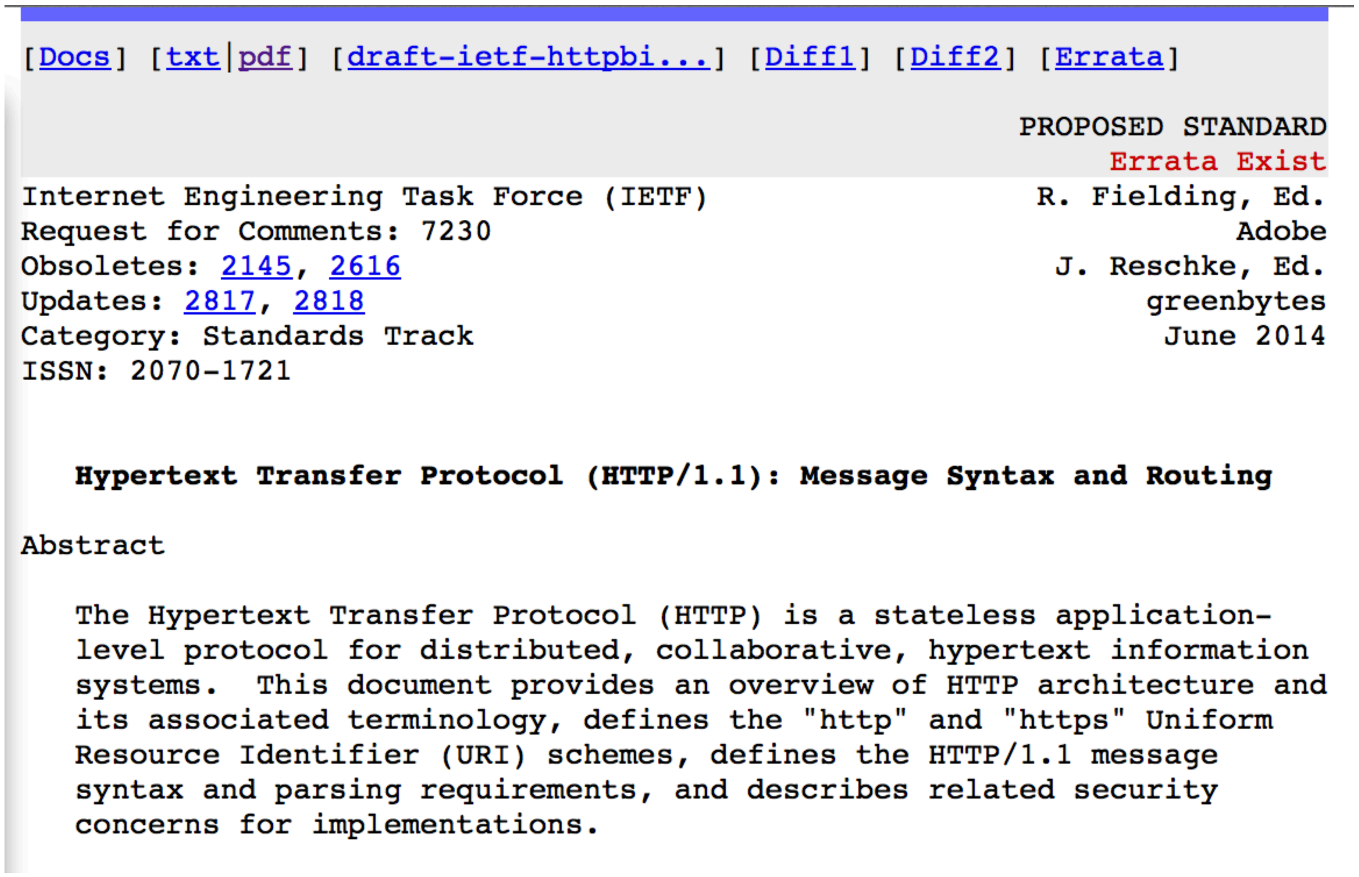
The genesis of the WWW

Source: Wikipedia

- Sir Tim Burners Lee
- Defined 3 standards:
  1. Hypertext Transfer Protocol (HTTP)
  2. Uniform Resource Identifiers (URI)
     - UR Locators (URL) – e.g. http://cmu.edu/heinz
       - *scheme://domain:port/path?query_string#fragment_id*
     - UR Names (URN) e.g. urn:isbn:0132143011
       - urn:*namespace_identifier:namespace_specific_string*
  3. Hypertext Markup Language (HTML)
- Then implemented to these standards:
  - A server program
  - A browser program

Who maintains the standards?

- The Internet Engineering Task Force (IETF) of the Internet Society, in cooperation with the World Wide Web Consortium (W3C)
- Published at http://www.w3.org/Protocols/



```
[Docs] [txt|pdf] [draft-ietf-httpbi...] [Diff1] [Diff2] [Errata]

                                                        PROPOSED STANDARD
                                                             Errata Exist
Internet Engineering Task Force (IETF)                   R. Fielding, Ed.
Request for Comments: 7230                                          Adobe
Obsoletes: 2145, 2616                                    J. Reschke, Ed.
Updates: 2817, 2818                                            greenbytes
Category: Standards Track                                       June 2014
ISSN: 2070-1721


   Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing

Abstract

   The Hypertext Transfer Protocol (HTTP) is a stateless application-
   level protocol for distributed, collaborative, hypertext information
   systems.  This document provides an overview of HTTP architecture and
   its associated terminology, defines the "http" and "https" Uniform
   Resource Identifier (URI) schemes, defines the HTTP/1.1 message
   syntax and parsing requirements, and describes related security
   concerns for implementations.
```

HTTP Methods

- Most frequently used:
  - GET– retrieve information
  - HEAD – query if there is information to retrieve
  - POST – append or modify information
  - PUT – add information
  - DELETE – delete information
- When choosing which HTTP methods to implement, consider:
  - Safety
  - Idempotence
- And when/how to use each

Safe

- A safe request is one that does not change the state of the resource on the server
  - – Except for trivial changes such as logging, caching, or incrementing a visit counter.
- E.g. GET should be safe
  - – GET should only retrieve information from a source
  - – GET should not update that resource
- GET ***could*** be used to update a resource, but as the HTTP protocol is defined, it ***should not*** be used in that way.

Head

- Another ***safe*** HTTP Method
- Requests headers only, but not content
- “If I were to GET this
  - Does it exist?
  - What would be its size?”
- Why might you want to use Head?

Idempotence

- Multiple identical requests are the same as a single request
  - Analog in math: taking the absolute value of a number.
- Nice to know on a potentially unreliable Internet:
  - If the first request didn't work, just try again.
  - If the Google search never returned, just reload

Idempotent Example

- http://www.andrew.cmu.edu/course/95-702/
- HTTP GET
- Repeating operation will get not change result
  - E.g. Repeated GET of course syllabus will not change the syllabus.
  - The resource may change on its own, so repeating might not get the same response.
    - But it would not be the GET which is changing the resource.
  - E.g. cnn.com is frequently updated, but not as a result of the GET.

Idempotent Methods

- Safe methods are idempotent
  - Because you are not changing the resource
  - Therefore GET and HEAD are idempotent
- PUT and DELETE are also idempotent
  - PUT adds or wholly replaces a resource
    - Do this a second time identically, you still have the same results.
  - DELETE removes a resource
    - DELETE a resource for the second time does not change its state.

Failure Models

- If you adhere to implementing GET, PUT, and DELETE as idempotent, then your failure model becomes easy:
  - If request fails, try it again.

Non-idempotent example:

- Create a new assignment on Blackboard
- https://blackboard.andrew.cmu.edu/webapps/blackboard/assignments/instructor/proc_edit_assignment.jsp?course_id=_1088365_1&content_id=_349041_1
- HTTP POST
- Repeating operation will add another assignment.

POST

- POST is for situations neither safe nor idempotent
  - use POST to change the resource
  - If you POST twice, then the results can be different after each change.
- POST should be reserved for modifying or adding to (not replacing), an existing resource.
- You have seen POST sometimes when you hit "Purchase" on a web store.
  - If you hit "Purchase" again, or reload, you may be charged twice.

What method should be used?

- To create a calendar event?
- To retrieve a database record?
- To delete a calendar event?
- Add links to participants on a calendar event?

Review

Method	Purpose?	Safe?	Idempotent?
GET			
PUT			
DELETE			
HEAD			
POST			

Review

Method	Purpose?	Safe?	Idempotent?
GET	Retrieve a resource	Y	Y
PUT	Insert or replace a resource	N	Y
DELETE	Remove a resource	N	Y
HEAD	Get header information only of a resource	Y	Y
POST	Append to or modify a resource	N	N

Misuse

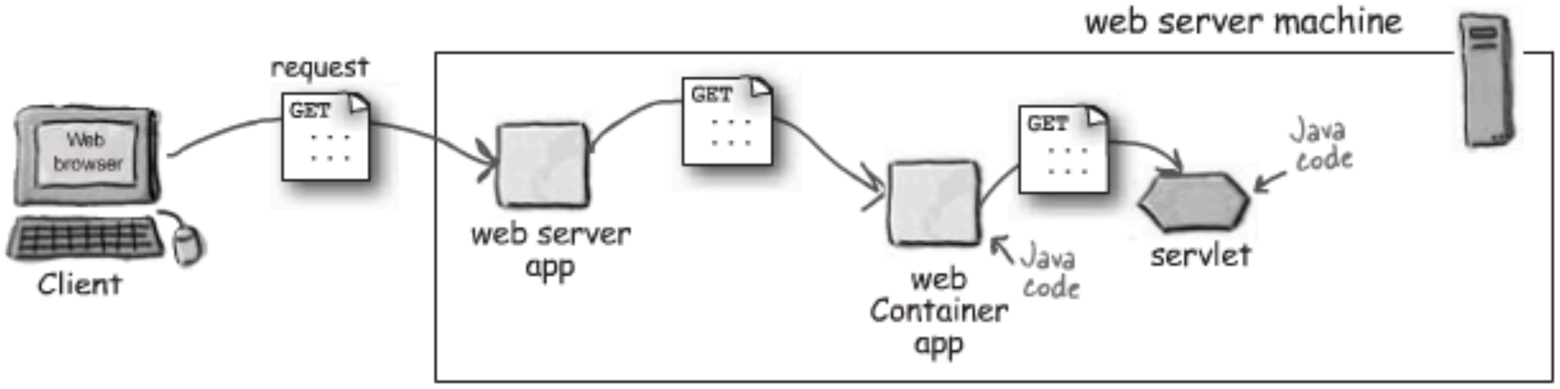
- Misuse is common
- E.g. Flickr uses GET to do everything, including removing a photo from your favorites list
  - http://api.flickr.com/services/rest/?method=flickr.favorites.remove&…

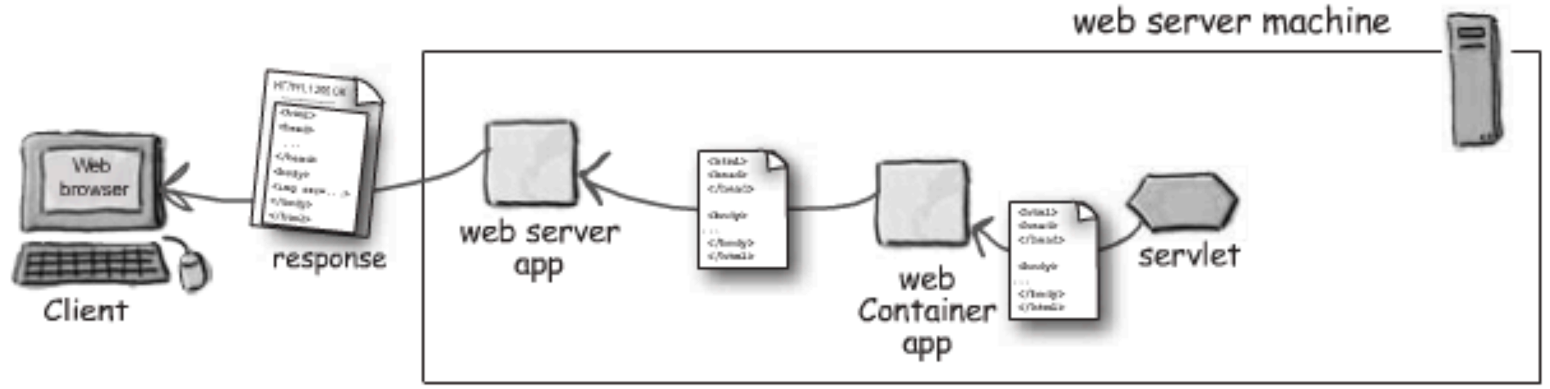
Java Servlets & JSP

- What does a server do when it receives an HTTP request?
- Java Enterprise Edition (JEE) is the platform (i.e. set of classes) for building distributed systems in Java.
- HttpServlet (commonly called a "Servlet") is the class designed to process HTTP requests.

- Great Reference:
  - Head First Servlets & JSP
    by Bryan Basham; Kathy Sierra; Bert Bates
  - Ch 1 – 3

Web applications - Overview



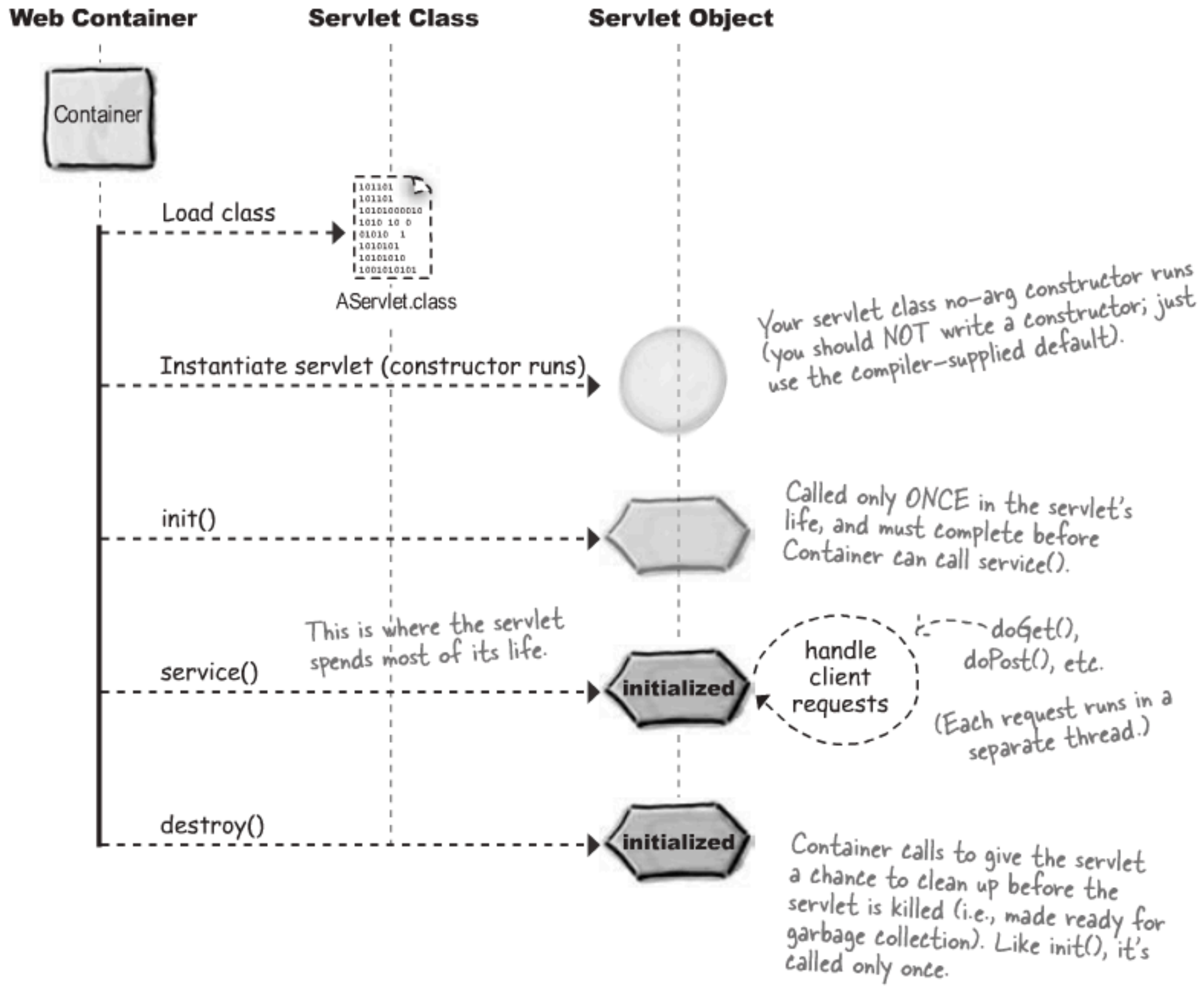


Source: Head First Servlets and JSP by Basham, Sierra, & Bates

What does the Web Container provide?

- Communication support
- Instantiating classes, passing arguments
- Multithreading (new thread for each request)
- Security

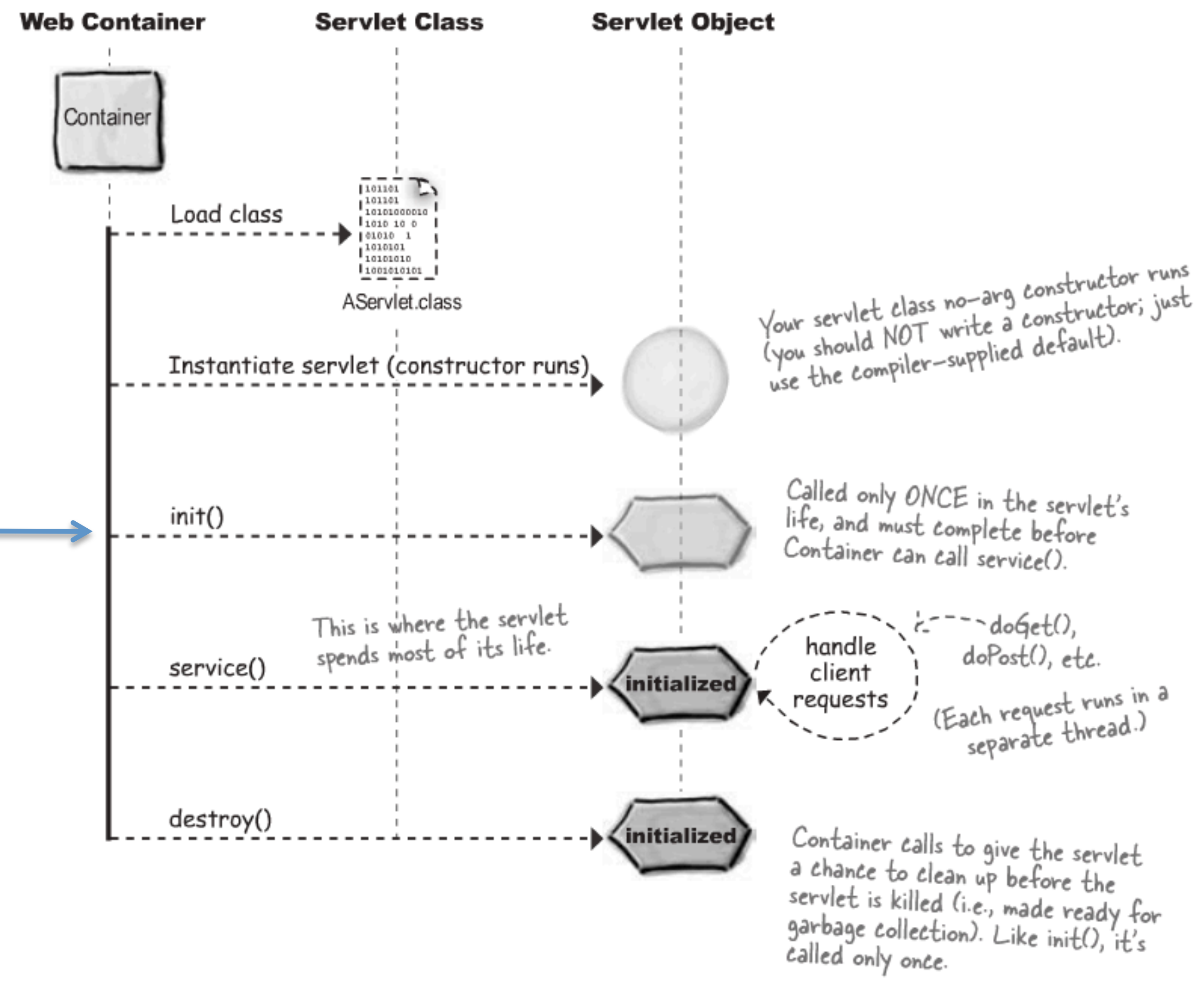
Servlet lifecycle


Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
service()
This is where the servlet spends most of its life.
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

Source: Head First Servlets and JSP by Basham, Sierra, & Bates

Servlet lifecycle

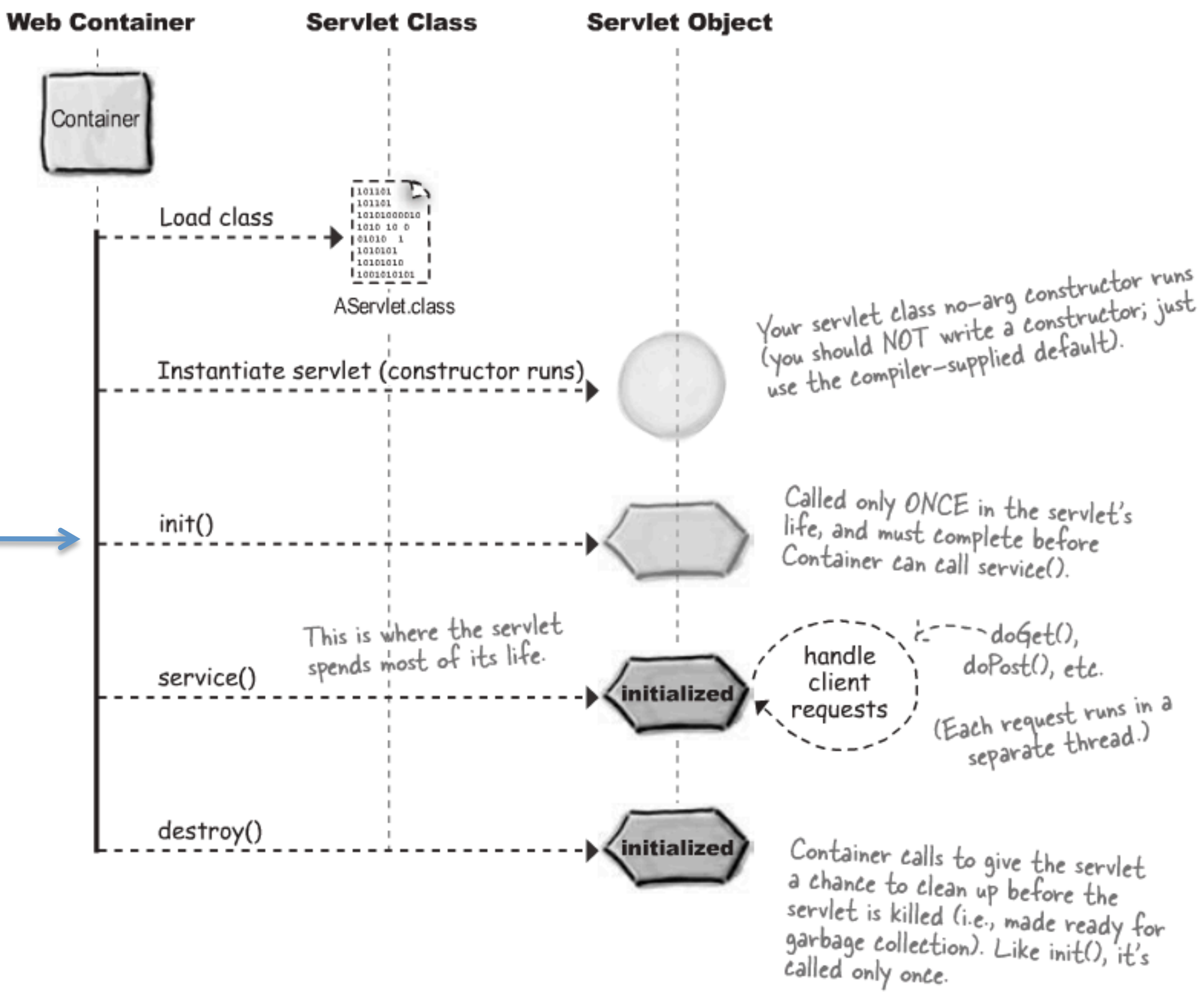
What might you do here?



Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
service()
This is where the servlet spends most of its life.
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

Source: Head First Servlets and JSP by Basham, Sierra, & Bates

Servlet lifecycle


Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
service()
This is where the servlet spends most of its life.
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

What might you do here?

- Initialize a database
- Open file(s)
- Register with other objects
- Connect (sockets) to other services

Source: Head First Servlets and JSP by Basham, Sierra, & Bates

Servlet lifecycle

You must overwrite one of:

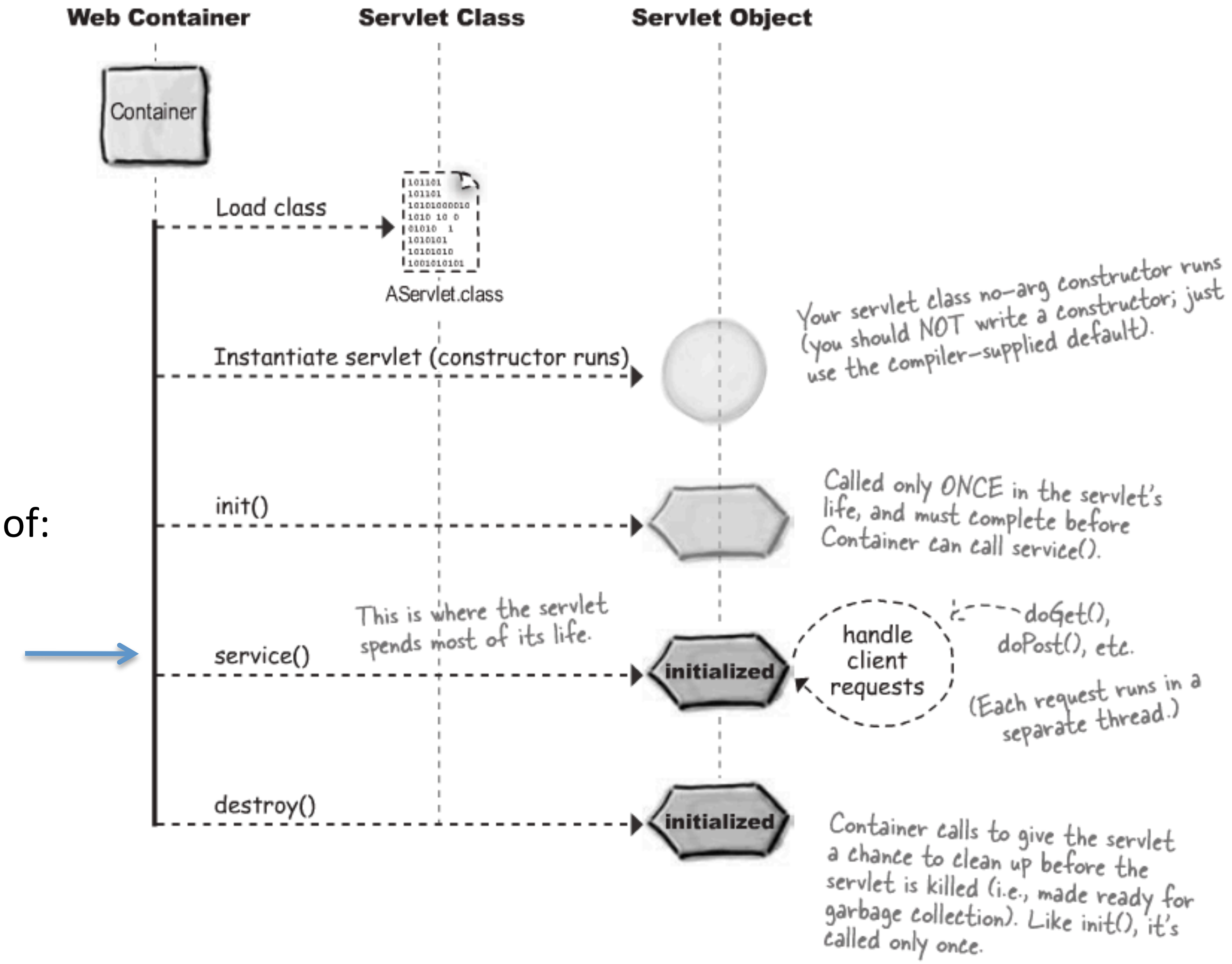
•doGet

•doPost

•doPut

•doDelete

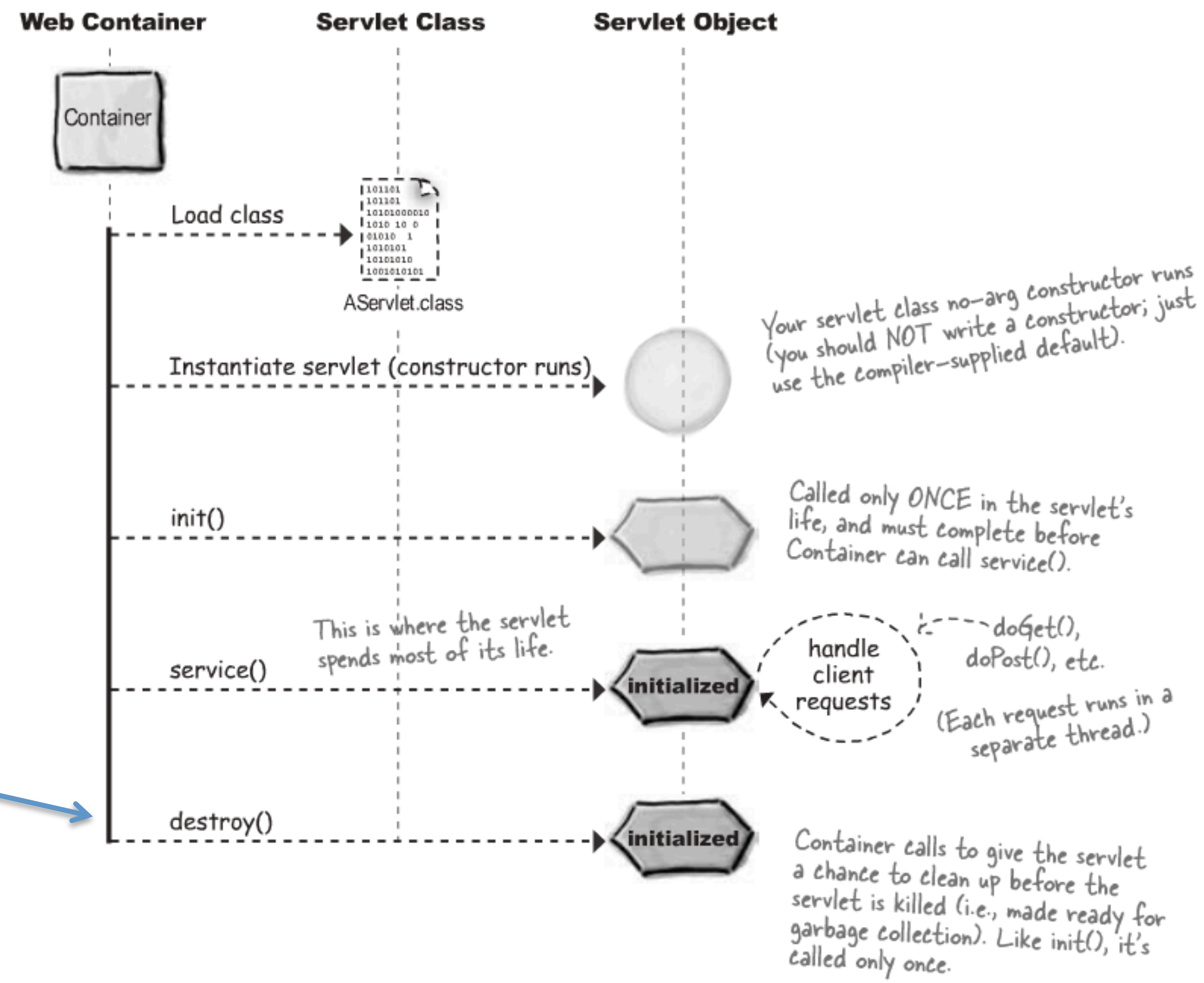
•doHead



Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
service()
This is where the servlet spends most of its life.
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

Source: Head First Servlets and JSP by Basham, Sierra, & Bates

Servlet lifecycle

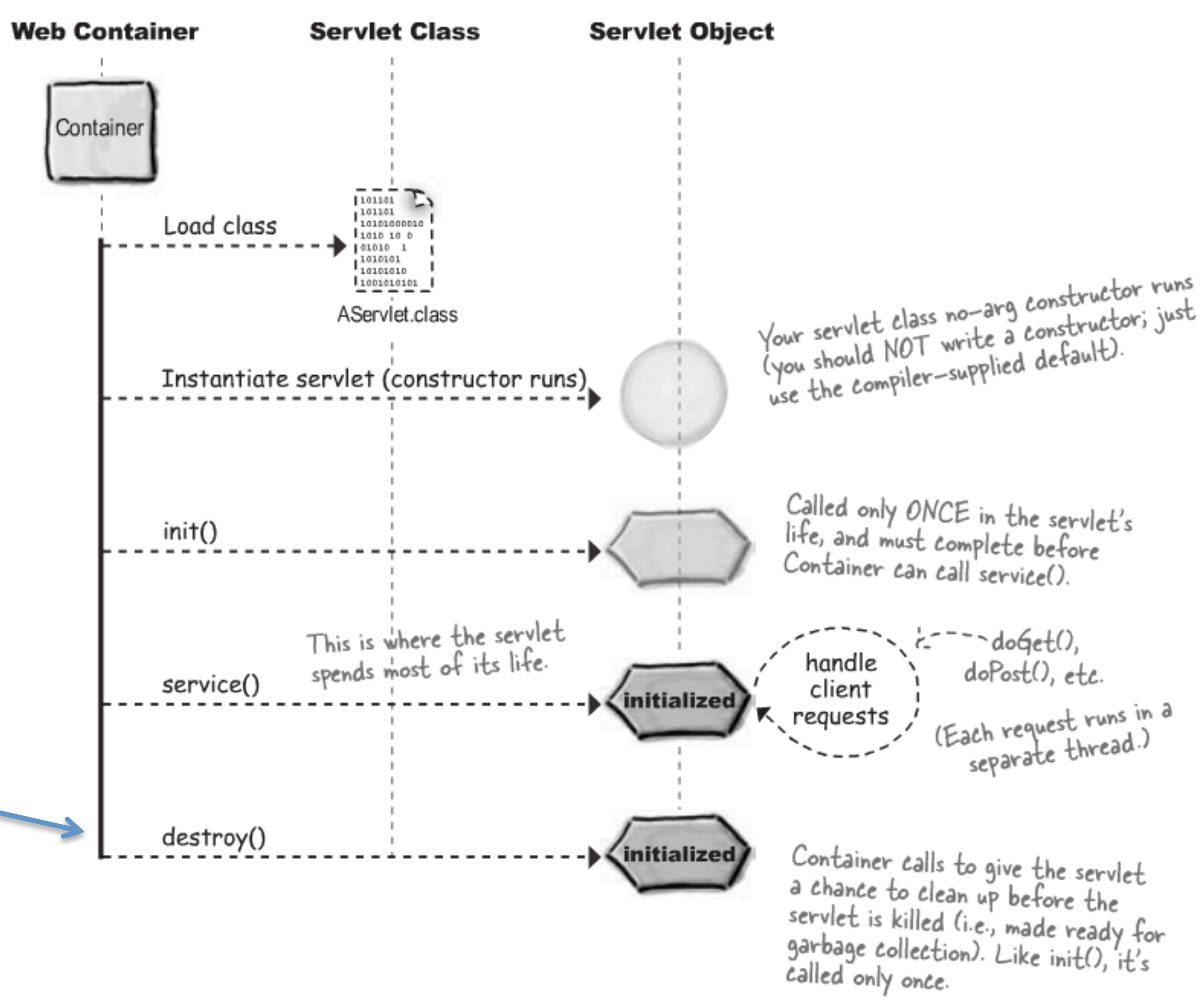


Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
This is where the servlet spends most of its life.
service()
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

What might you do here?

Source: Head First Servlets and JSP by Basham, Sierra, & Bates

Servlet lifecycle

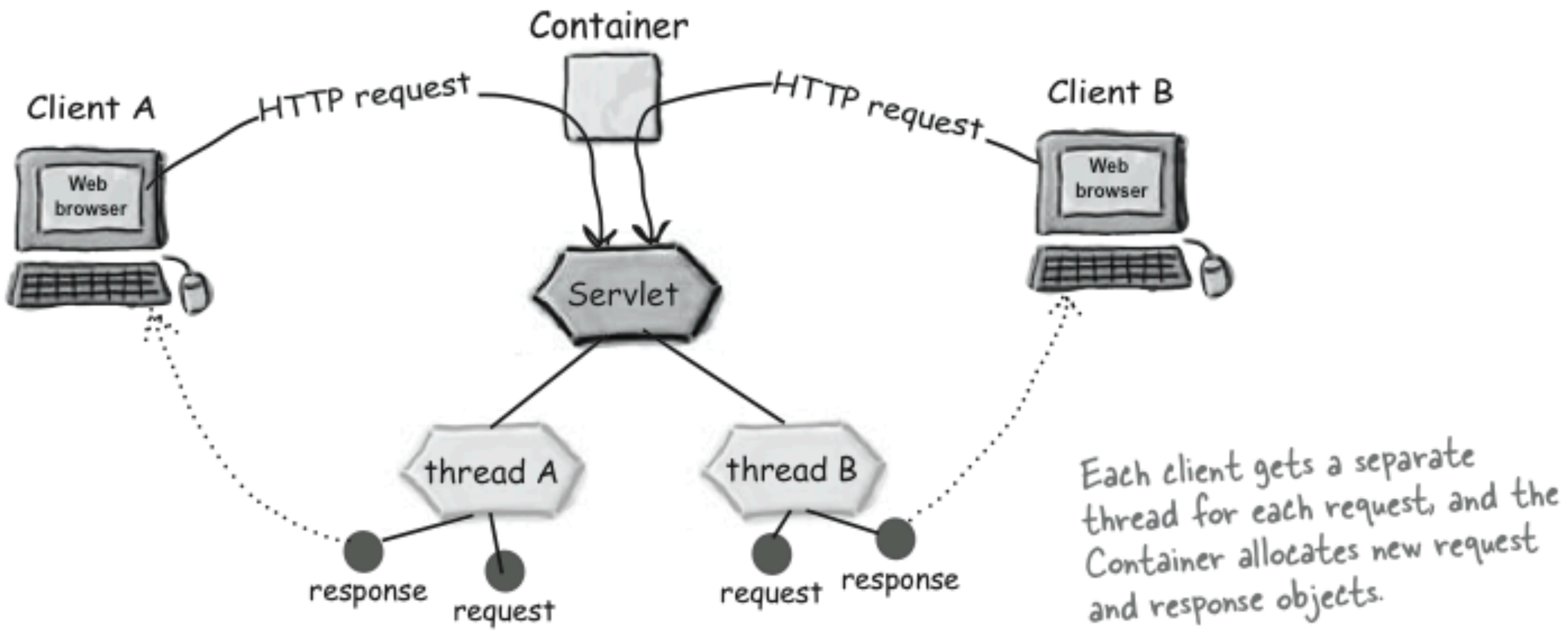

Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
service()
This is where the servlet spends most of its life.
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

What might you do here?

- Close a database or file
- Clean up relationships to other objects
- Close (sockets)

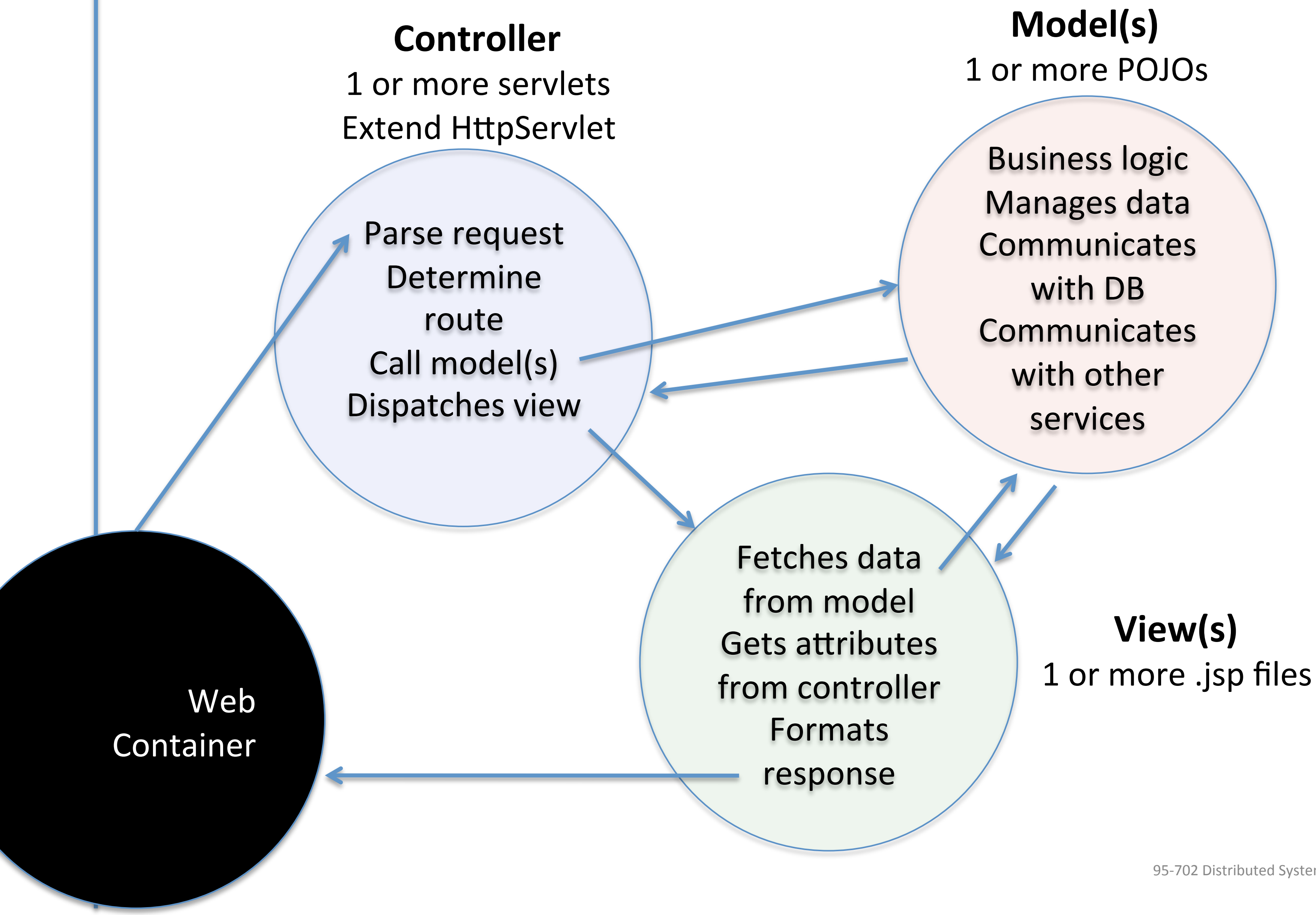
Source: Head First Servlets and JSP by Basham, Sierra, & Bates

One Servlet Object, Multiple Threads



Source: Head First Servlets and JSP by Basham, Sierra, & Bates

Model – View - Controller



Review InterestingPicture

- MVC Roles
- Servlet / Model interaction
- Model screen scraping
  - Show source being scraped
- Servlet / JSP interaction
- Parameters vs. Attributes
- Deployment Descriptor

HTML Resource

- If you have not used HTML before, you should learn the basics.
- W3Schools is a good HTML tutorial resource
  - Do the HTML Tutorial from start through the "HTML Attributes" page (6 pages)
    - http://www.w3schools.com/html
  - Do all the "HTML Forms" pages (4 pages)
    - http://www.w3schools.com/html/html_forms.asp
- Lynda is another good resource:
  - http://www.cmu.edu/lynda/

Cascading Style Sheets

- Cascading Style Sheets (CSS) provide web-page styling
- **We do not require any CSS in this class.**
- But you can add some if you like to make your small apps look better.
- W3Schools CSS Tutorial:
  – http://www.w3schools.com/css

JSP

- JSP = Java Server Pages
- Provides a higher-level abstraction to servlets
  - (JSP gets compiled into servlets)
- Predecessor to newer templating engines
  - E.g. erb (Ruby) and ejs (JavaScript)

JSP

- JSP is HTML, with Java code embedded in it.
- E.g. put 20 horizontal rule lines:

  (<hr> is the HTML for a single horizontal rule line)

  - <%   for (int x = 0; x< 20; x ++) { %> <hr> <% } %>
- JSP is used often to insert values received from the Controller or Model into the View
  - <%= .. %> is shorthand for "print"
  - E.g.
    - <img src=<%= request.getAttribute("pictureURL")%> >
- Good MVC dictates not to put logic, except for plugging in information, in the View

Lab vs Project Collaboration Rules

- Lab rules:
  - OK to talk to other students
  - OK to work together, and share hints and solutions
    - Please do!
  - Every student has to demo to a TA individually
- Project rules:
  - No talking to other students about code
  - No looking or discussing another student's code
  - No discussing or showing your code to others
  - Each student must work alone
  - Sharing code is cheating and results in failing course